Assignment 4, Section 1
Journaled File System

Rob Bowden

4/15/13

WARNING

There's a lot of "distribution" code for this pset that you are unfamiliar with.

THE DESIGN DOC IS ESPECIALLY IMPORTANT SO YOU DON'T DROWN (as always).

(Additional warning: there are many ways to write a journaled file system. This section has many "suggestions" for ways to make your lives easier)

Subsystems you will need to understand and modify for this assignment

- SFS
  - kern/fs/sfs/
  - user/sbin/mksfs/
  - user/sbin/sfsck/
- VFS
  - kern/vfs/
- Buffer Cache
  - kern/vfs/buf.c

What will your journal look like?

- More details in class tomorrow
- For now, understand that it is a *logical* journal, and that you only need to log *metadata*.
- It's just a place on your disk that keeps track of filesystem activity.
- You will need to figure out *what exactly* to log

When do you “bootstrap” your journal?

- On boot? Hmmm…

When do you “bootstrap” your journal?

- On boot? Hmmm…

- **You are allowed to assume only one log will be in use at any given time.**

When do you “bootstrap” your journal?

- On boot? Hmmm…
- **You are allowed to assume only one log will be in use at any given time.**
- **You may not (or at least should not) assume that this will be an SFS log**

When do you “bootstrap” your journal?

- On boot? Hmmm…
- **You are allowed to assume only one log will be in use at any given time.**
- **You may not (or at least should not) assume that this will be an SFS log**
- **You may not assume that the log will only be “in use” once during the kernel’s run. We could switch from SFS to some other filesystem.**

SFS vs VFS

- Where should your log go?

SFS vs VFS

- Where should your log go?
- Both!!

SFS vs VFS

- Where should your log go?
- Both!!
- VFS handles most things. The SFS portion of the log needs to “tell” the VFS how to construct the “records”, and what to do with them when replaying the log (more on these later and in class)

Other things you will need to change

- Buffer cache
- mksfs
- sfsck

Other things you will need to change

- Buffer cache
  - Need to keep track of *transactions*
- mksfs
- sfsck

Other things you will need to change

- Buffer cache
  - Need to keep track of *transactions*
- mksfs
  - Need to be sure the log region of the disk is zeroed out
- sfsck

Other things you will need to change

- Buffer cache
  - Need to keep track of *transactions*
- mksfs
  - Need to be sure the log region of the disk is zeroed out
- Sfsck
  - “Log Blocks” are now a thing, and need to be marked as “in use”

Transactions

- What are they?

Transactions

- What are they?

```
struct transaction {
    ???
}
```

Back to the buffer cache

- Can't write a buffer out until the transaction has completed
  - Must modify buffer cache to enforce this
- Can a single buffer be involved in multiple transactions?
- How does a transaction keep track of the buffers in use in order to "unmark" them?

Operations on Transactions

- begin
- commit
- abort

Operations on Transactions

- begin
  - Initializes a new transaction with a unique ID
- commit
- abort

Operations on Transactions

- begin
  - Initializes a new transaction with a unique ID
- Commit
  - Writes to the log on disk that the transaction committed
  - Unmarks all buffers that are “in use”
- abort

Operations on Transactions

- begin
  - Initializes a new transaction with a unique ID
  - Writes to disk?
- Commit
  - Writes to the log on disk that the transaction committed
  - Unmarks all buffers that are “in use”
- abort
  - Almost identical to commit, except writes that it aborted (maybe?)

What happens during a transaction?

- We touch buffers

What happens during a transaction?

- We touch buffers
  - (And keep track of them)

What happens during a transaction?

- We touch buffers
  - (And keep track of them)
- We touch metadata

What happens during a transaction?

- We touch buffers
  - (And keep track of them)
- We touch metadata
  - These changes go into the log… More on that to come

What happens during a transaction?

- We touch buffers
  - (And keep track of them)
- We touch metadata
  - These changes go into the log… More on that to come
- That's pretty much it!

So what sfs operations need to be logged?

- sfs_truncate
- sfs_mkdir
- sfs_remove
- …

- Notice the pattern?

How to log sfs_*

- Start the transaction at the “beginning” of the function
- Abort the transaction under error scenarios
- Commit the transaction
- Make sure any touched buffers are added to the transaction and any metadata-modifications added to the log

How do you write metadata modifications to the log?

- Write them as you make them?

How do you write metadata modifications to the log?

- Write them as you make them?
  - Less than ideal

How do you write metadata modifications to the log?

- Write them as you make them?
  - Less than ideal
    - May need to write many times over the course of a transaction

How do you write metadata modifications to the log?

- Write them as you make them?
  - Less than ideal
    - May need to write many times over the course of a transaction
    - Actually wouldn't need to keep track of buffers if we did this. Why?

How do you write metadata modifications to the log?

- Write them as you make them?
  - Less than ideal
    - May need to write many times over the course of a transaction
    - Actually wouldn't need to keep track of buffers if we did this. Why?
- Buffer them in memory!!

Two different logs

- In-memory and on-disk

Two different logs

- In-memory and on-disk
- During a transaction, when we touch metadata, we write it to the in-memory log.

Two different logs

- In-memory and on-disk
- During a transaction, when we touch metadata, we write it to the in-memory log
- At some point, we write the entire in-memory log to disk

Writing memory-log to disk

- When?

Writing memory-log to disk

- When?

1) When it starts getting too full

Writing memory-log to disk

- When?

1) When it starts getting too full

2) During a transaction commit or abort

Writing memory-log to disk

- When?

1) When it starts getting too full

2) During a transaction commit or abort
   - Less than ideal (a synchronous write), but what happens if we don't?

On-disk log

- Frequently depicted as a circular buffer

On-disk log

- Frequently depicted as a circular buffer
  - We wrap around and overwrite old transactions at the *beginning* of the buffer

On-disk log

- Frequently depicted as a circular buffer
  - We wrap around and overwrite old transactions at the *beginning* of the buffer
  - Need to be sure those metadata changes have been written to disk!!

On-disk log

- Frequently depicted as a circular buffer
  - We wrap around and overwrite old transactions at the *beginning* of the buffer
  - Need to be sure those metadata changes have been written to disk!!
- Even as a circular buffer, it can still fill up…

Checkpointing

- Processes everything in the log so we can "start over"
- Flush the in-memory log to the disk, and then process all of the blocks in the log.
- Multiple ways to represent "starting over": as a special "checkpoint" record in the log, or just zeroing out the entire log once you are done.

HINT

- Probably want some generic way to iterate over the entire log

HINT

- Probably want some generic way to iterate over the entire log
  - A function that takes a function pointer? Special macro?

HINT

- Probably want some generic way to iterate over the entire log
  - A function that takes a function pointer? Special macro?
- Don't need a circular buffer to get things working

Remember that there are two separate but related tasks

- Logging things while the filesystem is live

- Replaying the log

Logging things while the filesystem is live

- Do this part *FIRST*.
- GET IT WORKING, or you have no way to test the second part.
- After you think you've implemented things, write functions that will iterate over your log, printing out its contents.
- The printing functions don't even have to be in C! You can open up your disk as a regular file, and read through it.

Replaying the Log

- When should this happen?

Replaying the Log

- When should this happen?
- On bootstrap?

Replaying the Log

- When should this happen?
- On bootstrap?
  - But what if we didn't crash last time?

Replaying the Log

- When should this happen?
- On bootstrap?
  - But what if we didn't crash last time?
- What do we need to replay?

Replaying the Log

- When should this happen?
- On bootstrap?
  - But what if we didn't crash last time?
- What do we need to replay?
  - Undo logging vs redo logging

Synchronization

- Two threads logging something at the same time

Synchronization

- Two threads logging something at the same time
  - Can they be a part of the same transaction?

Synchronization

- Two threads logging something at the same time
  - Can they be a part of the same transaction?
- A thread is currently checkpointing, and another wants to log something

Synchronization

- Two threads logging something at the same time
  - Can they be a part of the same transaction?
- A thread is currently checkpointing, and another wants to begin a transaction
- A thread is currently in a transaction, but another wants to checkpoint

Order of Importance

1) No corruption of metadata

2) No leaking of erased user data

3) Other deeper user-data consistency issues